

Storypoint Problem - mule

September 6, 2024

1 Storypoint Prediction: Regression Approach

1.1 Preparation

```
[ ]: import os
import json

import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
import seaborn as sns
from scipy.sparse import csr_matrix, hstack, vstack

from sklearn.pipeline import Pipeline
from sklearn.preprocessing import RobustScaler
from sklearn.metrics import mean_squared_error, r2_score, mean_absolute_error, f1_score, precision_score, recall_score, accuracy_score
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.model_selection import cross_val_score, learning_curve, validation_curve
from trainer import GridSearchCVTrainer

project_name = 'mule'
```

1.1.1 Plot learning curve

```
[ ]: def plot_learning_curve(estimator, title, X, y, ylim=None, cv=None,
                             n_jobs=-1, train_sizes=np.linspace(.1, 1.0, 10)):
    plt.figure()          # Create new figure
    plt.title(title)      # Set title of the figure

    # Set y-axis limits: ylim=(min, max)
    if ylim is not None:
        plt.ylim(*ylim)

    plt.xlabel("Training examples") # Set x-axis label
    plt.ylabel("Score")             # Set y-axis label
```

```

# Generate learning curve data
train_sizes, train_scores, test_scores = learning_curve(
    estimator, X, y, cv=cv, n_jobs=n_jobs, train_sizes=train_sizes,
    ↪scoring='neg_mean_squared_error')

train_scores_mean = np.mean(train_scores, axis=1) # Calculate mean of
↪training scores
train_scores_std = np.std(train_scores, axis=1) # Calculate standard
↪deviation of training scores
test_scores_mean = np.mean(test_scores, axis=1) # Calculate mean of test
↪scores
test_scores_std = np.std(test_scores, axis=1) # Calculate standard
↪deviation of test scores

plt.grid() # Display grid

# Fill the area between the mean training score and the mean +/- std
↪training score
plt.fill_between(train_sizes, train_scores_mean - train_scores_std,
                 train_scores_mean + train_scores_std, alpha=0.1,
                 color="r")

# Fill the area between the mean test score and the mean +/- std test score
plt.fill_between(train_sizes, test_scores_mean - test_scores_std,
                 test_scores_mean + test_scores_std, alpha=0.1, color="g")

# Plot mean training score as points
plt.plot(train_sizes, train_scores_mean, 'o-', color="r",
         label="Training score")
# Plot mean test score as points
plt.plot(train_sizes, test_scores_mean, 'o-', color="g",
         label="Validation score")

plt.legend(loc="best") # Display legend
return plt

```

1.1.2 Plot validation curve

```

[ ]: def plot_validation_curve(estimator, title, X, y, param_name, param_range,
                             y_lim=None, cv=10, n_jobs=-1):
    train_scores, val_scores = validation_curve(estimator=estimator, X=X, y=y,
                                                param_name=param_name,
        ↪param_range=param_range,
                                                cv=cv, n_jobs=n_jobs,
        ↪scoring='neg_mean_squared_error')

```

```

# Calculate mean and standard deviation of training and validation scores
train_mean = np.mean(train_scores, axis=1)
tran_std = np.std(train_scores, axis=1)
val_mean = np.mean(val_scores, axis=1)
val_std = np.std(val_scores, axis=1)
print(val_mean)

# Plot train scores
plt.plot(param_range, train_mean, color='r', marker='o', markersize=5,
↪label='Training score')
plt.fill_between(param_range, train_mean + tran_std, train_mean - tran_std,
↪alpha=0.15, color='r')

# Plot validation scores
plt.plot(param_range, val_mean, color='g', linestyle='--', marker='s',
↪markersize=5, label='Validation score')
plt.fill_between(param_range, val_mean + val_std, val_mean - val_std,
↪alpha=0.15, color='g')

plt.title(title)          # Set title of the plot
plt.grid()                # Display grid
plt.xscale('log')          # Set x-axis scale to log
plt.legend(loc='best')    # Display legend
plt.xlabel('Parameter')   # Set x-axis label
plt.ylabel('Score')        # Set y-axis label

# Set y-axis limits
if y_lim != None:
    plt.ylim(y_lim)

return plt

```

1.1.3 Evaluate model

```

[ ]: def evaluate_model(model, model_name, X_test, y_test, y_logscale=False,
↪save_directory=None):
    y_pred = model.predict(X_test)
    if(y_logscale):
        y_pred = np.exp(y_pred)

    lines = [model_name + '\s evaluation results:']

    mse = mean_squared_error(y_test, y_pred)
    rmse = np.sqrt(mse)
    mae = mean_absolute_error(y_test, y_pred)
    r2 = r2_score(y_test, y_pred)

```

```

lines.append(f' - Mean squared error:      {mse:.2f}')
lines.append(f' - Root mean squared error: {rmse:.2f}')
lines.append(f' - Mean absolute error:     {mae:.2f}')
lines.append(f' - R2 error:                {r2:.2f}')

y_pred = np.round(y_pred).astype(int)
f1 = f1_score(y_test, y_pred, average='weighted')
precision = precision_score(y_test, y_pred, average='weighted',
↪zero_division=0)
recall = recall_score(y_test, y_pred, average='weighted', zero_division=0)
accuracy = accuracy_score(y_test, y_pred)

lines.append(f' - F1 score:                {f1:.2f}')
lines.append(f' - Precision:              {precision:.2f}')
lines.append(f' - Recall:                 {recall:.2f}')
lines.append(f' - Accuracy:               {accuracy:.2f}')
lines.append('-----')
lines.append('')

# Save to file

if(save_directory != None):
    filename = save_directory + project_name + '.txt'
    directory = os.path.dirname(filename)
    if not os.path.exists(directory):
        os.makedirs(directory)
    with open(filename, 'a') as f:
        for line in lines:
            print(line)
            f.write(line + '\n')
    else:
        for line in lines:
            print(line)

```

1.1.4 Set random seed

```

[ ]: import numpy as np
import random
import os

# Set random seed for numpy
np.random.seed(42)

# Set random seed for random
random.seed(42)

# Set random seed for os

```

```
os.environ['PYTHONHASHSEED'] = '42'
```

1.2 Dataset set-up

1.2.1 Bag of Words preprocessing

This is a Bag of Words preprocess approach. I will use 2 CountVectorizer from sklearn to change title and description to two 2 vectors and then concatenate them together. In the rest of this notebook, I will use cross-validation instead hold-out. Therefore, I will join the validation set with training set.

NOTE: I don't add the length of description (and the title) because it has a big deviation which may cause noise.

```
[ ]: # Import and remove NaN value

data_train = pd.concat([pd.read_csv('data/' + project_name + '/' + project_name_
    ↪ '_train.csv'),
                        pd.read_csv('data/' + project_name + '/' + project_name_
    ↪ '_valid.csv')])
data_test = pd.read_csv('data/' + project_name + '/' + project_name + '_test.
    ↪ csv')

data_train['description'].replace(np.nan, '', inplace=True)
data_test['description'].replace(np.nan, '', inplace=True)

# Vectorize title
title_vectorizer = CountVectorizer(ngram_range=(1, 2), min_df=2)
title_vectorizer.fit(pd.concat([data_train['title'], data_test['title']]))

# Vectorize description
description_vectorizer = CountVectorizer(ngram_range=(1, 2), min_df=2)
description_vectorizer.fit(pd.concat([data_train['description'],
    ↪ data_test['description']]))

X_train = hstack([title_vectorizer.transform(data_train['title']).astype(float),
                  description_vectorizer.transform(data_train['description']).
    ↪ astype(float),
                  data_train['title'].apply(lambda x : len(x)).to_numpy().
    ↪ reshape(-1, 1),
                  data_train['description'].apply(lambda x : len(x)).to_numpy().
    ↪ reshape(-1, 1)
                  ])

y_train = data_train['storypoint'].to_numpy().astype(float)
```

```

X_test = hstack([title_vectorizer.transform(data_test['title']).astype(float),
                 description_vectorizer.transform(data_test['description']).
                 ↪astype(float),
                 data_test['title'].apply(lambda x : len(x)).to_numpy().
                 ↪reshape(-1, 1),
                 data_test['description'].apply(lambda x : len(x)).to_numpy().
                 ↪reshape(-1, 1)
                 ])

y_test = data_test['storypoint'].to_numpy().astype(float)

```

```

[ ]: print('Check training dataset\'shape:', X_train.shape, y_train.shape)
     print('Check testing dataset\'shape:', X_test.shape, y_test.shape)

```

Check training dataset'shape: (801, 5115) (801,)
 Check testing dataset'shape: (88, 5115) (88,)

I will use log-scale the label to get a normal distribution of it.

```

[ ]: y_train_log = np.log(y_train)

```

1.3 Model training

1.3.1 Linear Regressor

```

[ ]: from sklearn.linear_model import ElasticNet

```

Define params-grid:

```

[ ]: dict_param = {
     'alpha': [.0001, .001, .01, .1, 1, 10, 100, 1000, 10000],
     'l1_ratio': [.0, .2, .4, .6, .8, 1],
     'max_iter': [3000],
     'random_state': [42]
     }

```

```

[ ]: gridsearch = GridSearchCVTrainer(name='Elastic Net', model=ElasticNet(),
     ↪param_grid=dict_param, cv=5, n_jobs=5)
     gridsearch.load_if_exists()
     gridsearch.fit(X_train, y_train_log)

     elastic_model = gridsearch.best_estimator_
     elastic_model.fit(X_train, y_train_log)

```

Out [00:00, ?it/s]

```

[ ]: ElasticNet(alpha=0.01, l1_ratio=1, max_iter=3000, random_state=42)

```

```
[ ]: evaluate_model(elastic_model, 'Elastic Net model', X_test, y_test,
    ↪y_logscale=True, save_directory='results/BoW/')
```

Elastic Net model's evaluation results:

```
- Mean squared error:      12.49
- Root mean squared error: 3.53
- Mean absolute error:     2.77
- R2 error:                -0.01
- F1 score:                0.06
- Precision:               0.10
- Recall:                  0.08
- Accuracy:                0.08
```

```
[ ]: elastic_model.get_params()
```

```
[ ]: {'alpha': 0.01,
      'copy_X': True,
      'fit_intercept': True,
      'l1_ratio': 1,
      'max_iter': 3000,
      'positive': False,
      'precompute': False,
      'random_state': 42,
      'selection': 'cyclic',
      'tol': 0.0001,
      'warm_start': False}
```

1.3.2 Support Vector Regressor

```
[ ]: from sklearn.svm import SVR
```

```
[ ]: dict_param = {
      'C': [.0001, .001, .01, .1, 1, 10, 100, 1000, 10000],
      'gamma': np.logspace(-9, 3, 13),
      'kernel': ['rbf']
    }
```

```
[ ]: grid_search = GridSearchCVTrainer(name="Support Vector Regressor", model=SVR(),
    ↪param_grid=dict_param, cv=5, n_jobs=5)
grid_search.load_if_exists()
grid_search.fit(X_train, y_train_log)

svr_model = grid_search.best_estimator_
svr_model.fit(X_train, y_train_log)
```

Oit [00:00, ?it/s]

```
[ ]: SVR(C=10, gamma=0.1)
```

```
[ ]: evaluate_model(svr_model, 'SVR model', X_test, y_test, y_logscale=True,  
    ↪ save_directory='results/BoW/')
```

SVR model's evaluation results:

```
- Mean squared error:      14.52  
- Root mean squared error: 3.81  
- Mean absolute error:     2.80  
- R2 error:                -0.17  
- F1 score:                0.00  
- Precision:               0.00  
- Recall:                  0.00  
- Accuracy:                0.00
```

```
[ ]: svr_model.get_params()
```

```
[ ]: {'C': 10,  
      'cache_size': 200,  
      'coef0': 0.0,  
      'degree': 3,  
      'epsilon': 0.1,  
      'gamma': 0.1,  
      'kernel': 'rbf',  
      'max_iter': -1,  
      'shrinking': True,  
      'tol': 0.001,  
      'verbose': False}
```

1.3.3 Random Forest Regressor

```
[ ]: from sklearn.ensemble import RandomForestRegressor
```

```
[ ]: dict_param = {  
      'max_depth' : [1000, 2000, 5000],  
      'min_samples_split': [25, 200, 1000],  
      'min_samples_leaf': [1, 2, 3, 4],  
      'max_features': [50, 100, 200],  
      'n_estimators': [1024],  
      'random_state': [42]  
    }
```

```
[ ]: grid_search = GridSearchCVTrainer(name="Random Forest Regressor",  
                                     model=RandomForestRegressor(),  
                                     param_grid=dict_param, cv = 5, n_jobs=5)  
grid_search.load_if_exists()
```



```
grid_search.fit(X_train, y_train_log)

rfr_model = grid_search.best_estimator_
rfr_model.fit(X_train, y_train_log)
```

Out [00:00, ?it/s]

```
[ ]: RandomForestRegressor(max_depth=1000, max_features=100, min_samples_leaf=3,
                           min_samples_split=25, n_estimators=1024, random_state=42)
```

```
[ ]: evaluate_model(rfr_model, 'Random Forest model', X_test, y_test,
                    ↪y_logscale=True, save_directory='results/BoW/')
-----
```

Random Forest model's evaluation results:

```
- Mean squared error:      13.64
- Root mean squared error: 3.69
- Mean absolute error:     2.70
- R2 error:                -0.10
- F1 score:                0.11
- Precision:               0.16
- Recall:                  0.14
- Accuracy:                0.14
```

```
[ ]: rfr_model.get_params()
```

```
[ ]: {'bootstrap': True,
      'ccp_alpha': 0.0,
      'criterion': 'squared_error',
      'max_depth': 1000,
      'max_features': 100,
      'max_leaf_nodes': None,
      'max_samples': None,
      'min_impurity_decrease': 0.0,
      'min_samples_leaf': 3,
      'min_samples_split': 25,
      'min_weight_fraction_leaf': 0.0,
      'monotonic_cst': None,
      'n_estimators': 1024,
      'n_jobs': None,
      'oob_score': False,
      'random_state': 42,
      'verbose': 0,
      'warm_start': False}
```

1.3.4 XGBoost

```
[ ]: from xgboost import XGBRegressor

[ ]: dict_param = {
    'eta' : np.linspace(0.01, 0.2, 3),
    'gamma': np.logspace(-2, 2, 5),
    'max_depth': np.asarray([3, 5, 7, 9]).tolist(),
    'min_child_weight': np.logspace(-2, 2, 5),
    'subsample': np.asarray([0.5, .1]),
    'reg_alpha': np.asarray([0.0, 0.05]),
    'n_estimators': np.asarray([10, 20, 50, 100]).tolist(),
    'random_state': [42]
}

[ ]: grid_search = GridSearchCVTrainer(name='XGBoost_
↳Regressor',model=XGBRegressor(), param_grid=dict_param, cv = 5, n_jobs=5)
grid_search.load_if_exists()
grid_search.fit(X_train, y_train_log)

xgb_model = grid_search.best_estimator_
xgb_model.fit(X_train, y_train_log)

Oit [00:00, ?it/s]

[ ]: XGBRegressor(base_score=None, booster=None, callbacks=None,
    colsample_bylevel=None, colsample_bynode=None,
    colsample_bytree=None, device=None, early_stopping_rounds=None,
    enable_categorical=False, eta=0.105, eval_metric=None,
    feature_types=None, gamma=0.1, grow_policy=None,
    importance_type=None, interaction_constraints=None,
    learning_rate=None, max_bin=None, max_cat_threshold=None,
    max_cat_to_onehot=None, max_delta_step=None, max_depth=5,
    max_leaves=None, min_child_weight=10.0, missing=nan,
    monotone_constraints=None, multi_strategy=None, n_estimators=20,
    n_jobs=None, num_parallel_tree=None, ...)

[ ]: evaluate_model(xgb_model, 'XGBoost Regressor model', X_test, y_test,
↳y_logscale=True, save_directory='results/BoW/')
```

XGBoost Regressor model's evaluation results:

- Mean squared error:	14.14
- Root mean squared error:	3.76
- Mean absolute error:	2.78
- R2 error:	-0.14
- F1 score:	0.07
- Precision:	0.11
- Recall:	0.08
- Accuracy:	0.08

```
[ ]: xgb_model.get_params()
```

```
[ ]: {'objective': 'reg:squarederror',
      'base_score': None,
      'booster': None,
      'callbacks': None,
      'colsample_bylevel': None,
      'colsample_bynode': None,
      'colsample_bytree': None,
      'device': None,
      'early_stopping_rounds': None,
      'enable_categorical': False,
      'eval_metric': None,
      'feature_types': None,
      'gamma': 0.1,
      'grow_policy': None,
      'importance_type': None,
      'interaction_constraints': None,
      'learning_rate': None,
      'max_bin': None,
      'max_cat_threshold': None,
      'max_cat_to_onehot': None,
      'max_delta_step': None,
      'max_depth': 5,
      'max_leaves': None,
      'min_child_weight': 10.0,
      'missing': nan,
      'monotone_constraints': None,
      'multi_strategy': None,
      'n_estimators': 20,
      'n_jobs': None,
      'num_parallel_tree': None,
      'random_state': 42,
      'reg_alpha': 0.0,
      'reg_lambda': None,
      'sampling_method': None,
      'scale_pos_weight': None,
      'subsample': 0.5,
      'tree_method': None,
      'validate_parameters': None,
      'verbosity': None,
      'eta': 0.105}
```

1.3.5 LightGBM

```
[ ]: from lightgbm import LGBMRegressor
     from sklearn.model_selection import ParameterSampler
```

```
[ ]: dict_param = {
    'n_estimator': [10, 20, 50, 100, 200, 500],
    'max_depth': np.asarray([5, 7, 9, 11, 13]).tolist(),
    'num_leaves': ((np.power(2, np.asarray([5, 7, 9, 11, 13]))) - 1) * (0.55 +
↪ (0.65 - 0.55) * np.random.rand(5))).astype(int).tolist(),
    'min_data_in_leaf': np.linspace(100, 1000, 4).astype(int).tolist(),
    'feature_fraction': np.linspace(0.6, 1, 3),
    'bagging_fraction': np.linspace(0.6, 1, 3),
    'learning_rate': [0.01],
    'verbose': [-1],
    'random_state': [42]
}

def custom_sampler(param_grid):
    for params in ParameterSampler(param_grid, n_iter=1e9):
        range_num_leaves = ((0.5 * (2**params['max_depth'] - 1)), (0.7 *
↪ (2**params['max_depth'] - 1)))
        if (range_num_leaves[0] <= params['num_leaves'] <= range_num_leaves[1]):
            for key, value in params.items():
                params[key] = [value]
            yield params
```

```
[ ]: grid_search = GridSearchCVTrainer(name='LightGBM Regressor',
↪ model=LGBMRegressor(),
                                     param_grid=list(custom_sampler(dict_param)), cv
↪ = 5, n_jobs=2)
grid_search.load_if_exists()
grid_search.fit(X_train, y_train_log)

lgbmr_model = grid_search.best_estimator_
lgbmr_model.fit(X_train, y_train_log)
```

c:\Users\aupho\AppData\Local\Programs\Python\Python311\Lib\site-packages\sklearn\model_selection_search.py:320: UserWarning: The total space of parameters 5400 is smaller than n_iter=1000000000. Running 5400 iterations. For exhaustive searches, use GridSearchCV.

```
warnings.warn(
Oit [00:00, ?it/s]
```

```
[ ]: LGBMRegressor(bagging_fraction=0.6, feature_fraction=0.6, learning_rate=0.01,
                  max_depth=5, min_data_in_leaf=100, n_estimator=10, num_leaves=19,
                  random_state=42, verbose=-1)
```

```
[ ]: evaluate_model(lgbmr_model, 'LightGBM regressor model', X_test, y_test,
↳y_logscale=True, save_directory='results/BoW/')
```

LightGBM regressor model's evaluation results:

```
- Mean squared error:      13.82
- Root mean squared error: 3.72
- Mean absolute error:     2.74
- R2 error:                -0.11
- F1 score:                0.03
- Precision:               0.13
- Recall:                  0.02
- Accuracy:                0.02
```

c:\Users\aupho\AppData\Local\Programs\Python\Python311\Lib\site-packages\lightgbm\basic.py:1218: UserWarning: Converting data to scipy sparse matrix.

```
_log_warning("Converting data to scipy sparse matrix.")
```

```
[ ]: lgbmr_model.get_params()
```

```
[ ]: {'boosting_type': 'gbdt',
      'class_weight': None,
      'colsample_bytree': 1.0,
      'importance_type': 'split',
      'learning_rate': 0.01,
      'max_depth': 5,
      'min_child_samples': 20,
      'min_child_weight': 0.001,
      'min_split_gain': 0.0,
      'n_estimators': 100,
      'n_jobs': None,
      'num_leaves': 19,
      'objective': None,
      'random_state': 42,
      'reg_alpha': 0.0,
      'reg_lambda': 0.0,
      'subsample': 1.0,
      'subsample_for_bin': 200000,
      'subsample_freq': 0,
      'verbose': -1,
      'n_estimator': 10,
      'min_data_in_leaf': 100,
      'feature_fraction': 0.6,
      'bagging_fraction': 0.6}
```

1.3.6 Stacked model:

```
[ ]: from mlxtend.regressor import StackingCVRegressor
```

Define component models:

```
[ ]: directory = 'trainer_checkpoint_data/'
with open(directory + 'elastic net_checkpoint.json', 'r') as json_file:
    data = json.load(json_file)
    elastic_model = ElasticNet(**data['best_params'])

with open(directory + 'support vector regressor_checkpoint.json', 'r') as json_file:
    data = json.load(json_file)
    svr_model = SVR(**data['best_params'])

with open(directory + 'random forest regressor_checkpoint.json', 'r') as json_file:
    data = json.load(json_file)
    rfr_model = RandomForestRegressor(**data['best_params'], n_jobs=-1)

with open(directory + 'xgboost regressor_checkpoint.json', 'r') as json_file:
    data = json.load(json_file)
    xgb_model = XGBRegressor(**data['best_params'])

with open(directory + 'lightgbm regressor_checkpoint.json', 'r') as json_file:
    data = json.load(json_file)
    lgbmr_model = LGBMRegressor(**data['best_params'])
```

Define blended model:

```
[ ]: stack_gen = StackingCVRegressor(regressors=(xgb_model, lgbmr_model, svr_model,
    elastic_model, rfr_model),
    meta_regressor=elastic_model,
    use_features_in_secondary=True, n_jobs=-1,
    random_state=42)
stack_gen.fit(X_train, y_train_log)
```

```
[ ]: StackingCVRegressor(meta_regressor=ElasticNet(alpha=0.01, l1_ratio=1,
    max_iter=3000, random_state=42),
    n_jobs=-1, random_state=42,
    regressors=(XGBRegressor(base_score=None, booster=None,
    callbacks=None,
    colsample_bylevel=None,
    colsample_bynode=None,
    colsample_bytree=None, device=None,
    early_stopping_rounds=None,
    enable_categorical=False,
```

```

        eta=0.105, eval_metric=Non...
        learning_rate=0.01, max_depth=5,
        min_data_in_leaf=100,
        n_estimator=10, num_leaves=19,
        random_state=42, verbose=-1),
SVR(C=10, gamma=0.1),
ElasticNet(alpha=0.01, l1_ratio=1,
            max_iter=3000, random_state=42),
RandomForestRegressor(max_depth=1000,
                       max_features=100,
                       min_samples_leaf=3,
                       min_samples_split=25,
                       n_estimators=1024,
                       n_jobs=-1,
                       random_state=42)),
        use_features_in_secondary=True)

```

```

[ ]: evaluate_model(stack_gen, 'Stacking model', X_test, y_test, y_logscale=True,
    ↪ save_directory='results/BoW/')

```

Stacking model's evaluation results:

```

- Mean squared error:      12.39
- Root mean squared error: 3.52
- Mean absolute error:     2.70
- R2 error:                0.00
- F1 score:                0.10
- Precision:               0.17
- Recall:                  0.11
- Accuracy:                0.11

```

c:\Users\aupho\AppData\Local\Programs\Python\Python311\Lib\site-packages\lightgbm\basic.py:1218: UserWarning: Converting data to scipy sparse matrix.

```

    _log_warning("Converting data to scipy sparse matrix.")

```